INTRODUCTION TO HADOOP

Prepared By : Manoj Kumar Joshi & Vikas Sawhney

General Agenda

Introduction to Hadoop Architecture

Acknowledgement

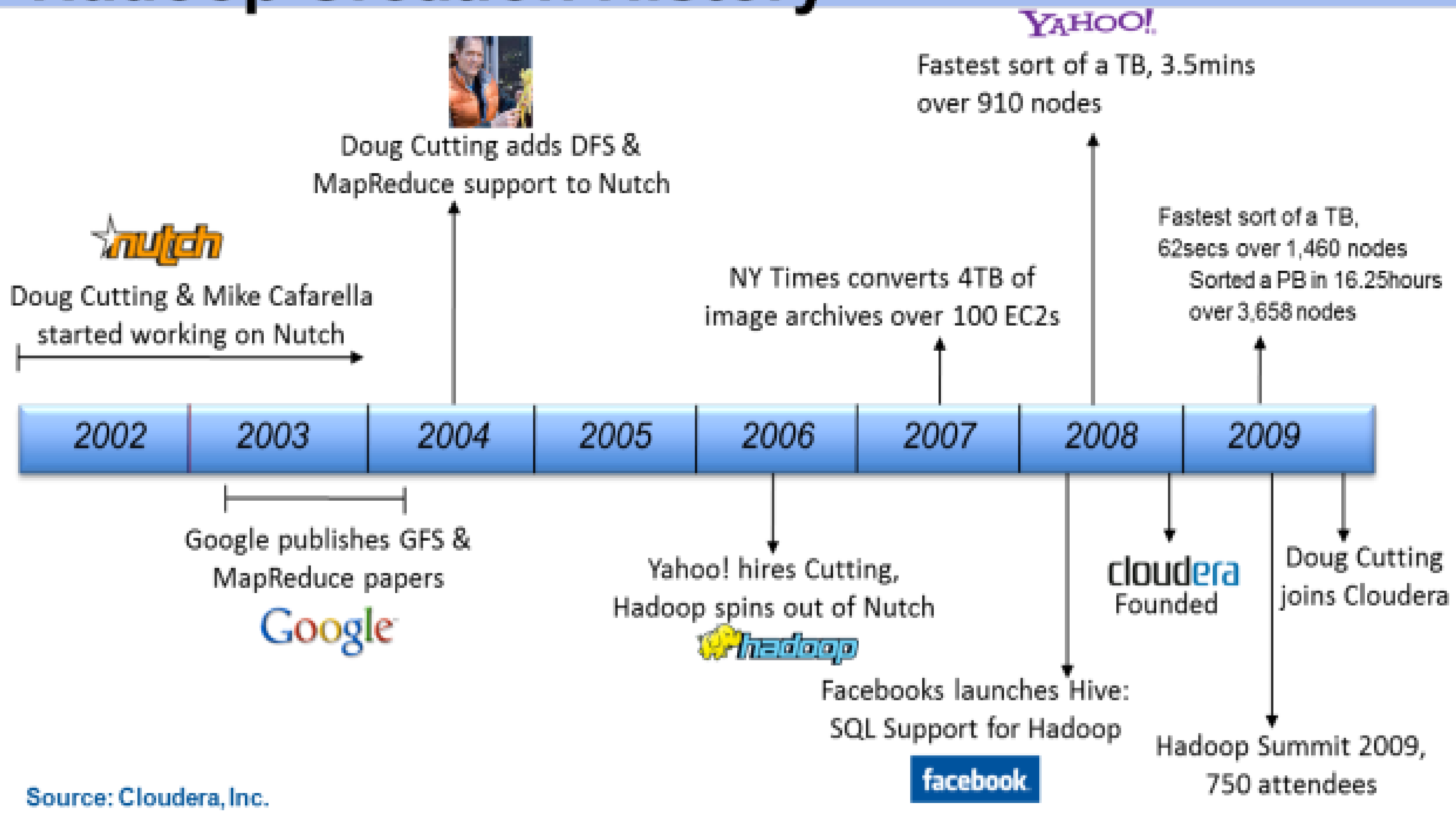
- Thanks to all the authors who left their self-explanatory images on the internet.
- Thanks to bradhedlund.com and Cloudera Inc for their blogs on Hadoop
- We own the errors of course

History Of Hadoop

- Hadoop was started by Doug Cutting to support two of his other well known projects, Lucene and Nutch
- Hadoop has been inspired by Google's File System (GFS) which was detailed in a paper by released by Google in 2003
- Hadoop, originally called Nutch Distributed File System (NDFS) split from Nutch in 2006 to become a sub-project of Lucene. At this point it was renamed to Hadoop.

History Of Hadoop (Contd.)

Hadoop Creation History



Doug Cutting & Mike Cafarella started working on Nutch

Google publishes GFS & MapReduce papers

Doug Cutting adds DFS & MapReduce support to Nutch

Yahoo! hires Cutting, Hadoop spins out of Nutch

NY Times converts 4TB of image archives over 100 EC2s

Facebooks launches Hive: SQL Support for Hadoop

Fastest sort of a TB, 3.5mins over 910 nodes

Cloudera Founded

Fastest sort of a TB, 62secs over 1,460 nodes

Sorted a PB in 16.25hours over 3,658 nodes

Hadoop Summit 2009, 750 attendees

Doug Cutting joins Cloudera

Source: Cloudera, Inc.

Distributed File System (DFS)

- A **Distributed File System** (DFS) is simply a classical model of a file system distributed across multiple machines. The purpose is to promote sharing of dispersed files.
- This is an area of active research interest today.
- Clients should view a DFS the same way they would a centralized FS; the distribution is hidden at a lower level.
- A DFS provides high throughput data access and fault tolerance.

Why Hadoop ?

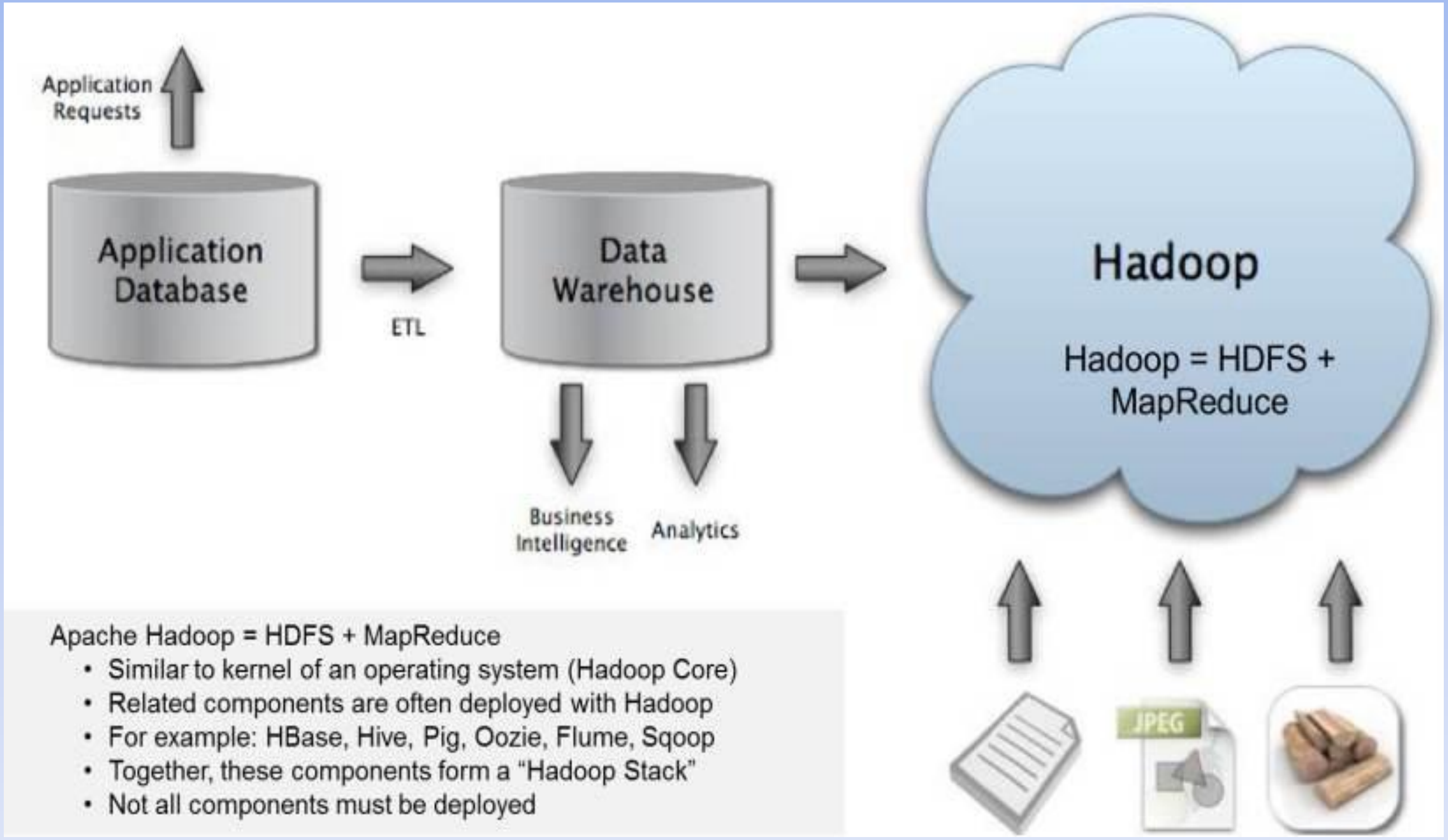
Hadoop infrastructure provides these capabilities

- Scalability
  - -Thousands of Compute Nodes
  - -Petabytes of data
- Cost effective
  - -Runs On Low Cost Commodity Hardware
- Efficient
  - -By distributing the data, Hadoop can process it in parallel on the nodes where the data is located.

What Is Hadoop ?

- Open source software platform for scalable, distributed computing
- Hadoop provides fast and reliable analysis of both structured data and unstructured data
- Apache Hadoop software library is essentially a framework that allows for the distributed processing of large datasets across clusters of computers using a simple programming model.
- Hadoop can scale up from single servers to thousands of machines, each offering local computation and storage.

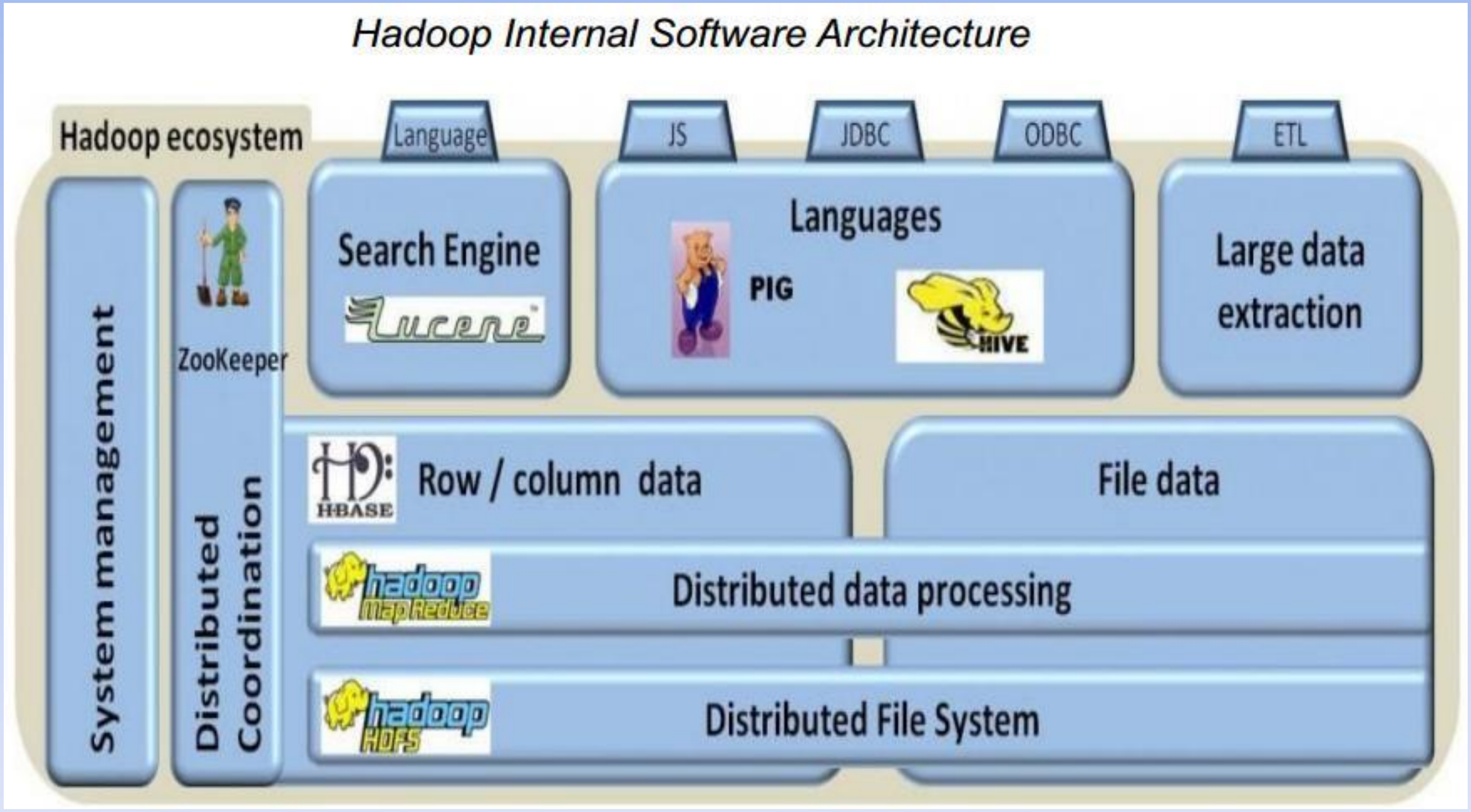
Hadoop Architecture



Apache Hadoop = HDFS + MapReduce

- Similar to kernel of an operating system (Hadoop Core)
- Related components are often deployed with Hadoop
- For example: HBase, Hive, Pig, Oozie, Flume, Sqoop
- Together, these components form a “Hadoop Stack”
- Not all components must be deployed

Hadoop Stack



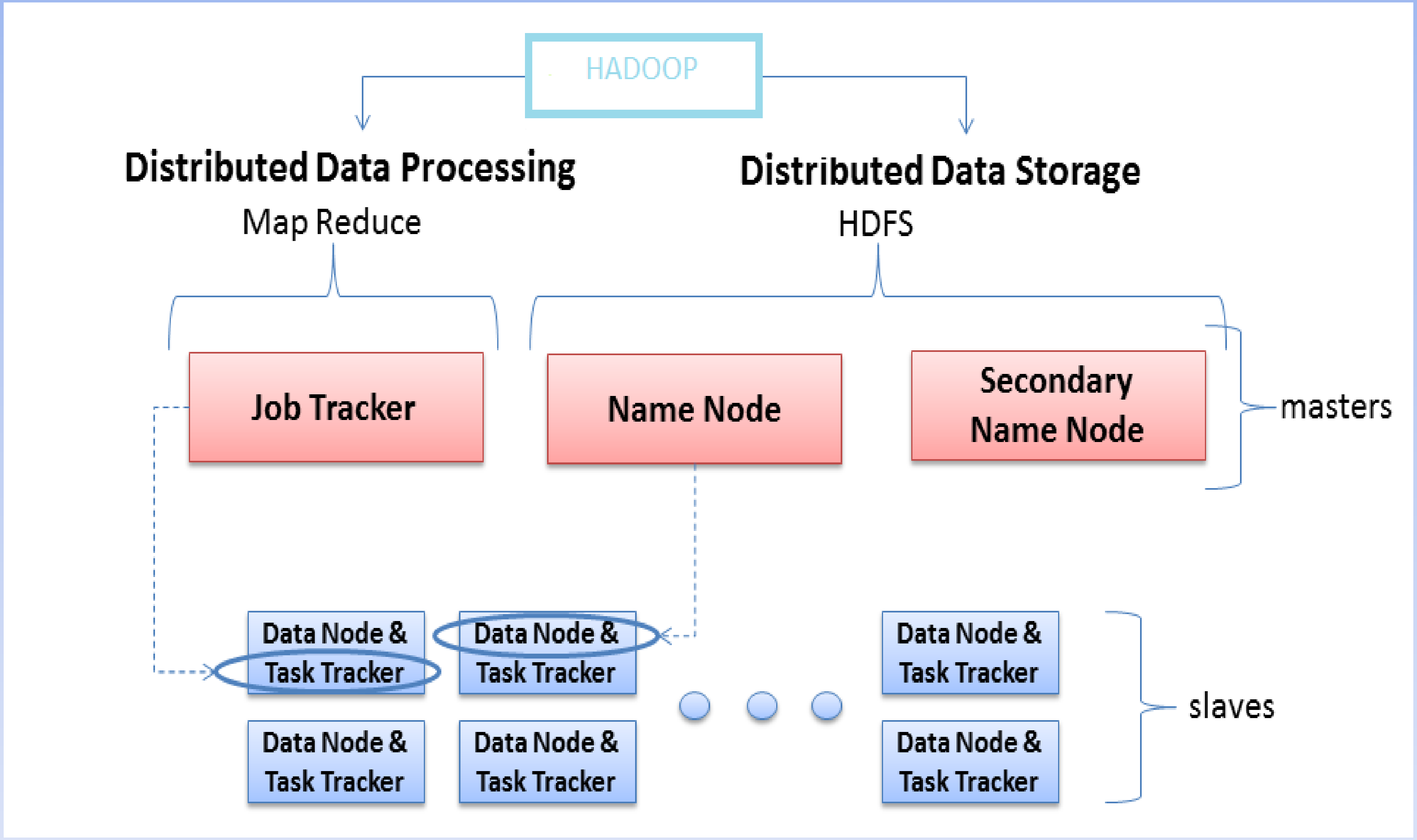
Hadoop Ecosystems Projects

- Hadoop Ecosystem Projects includes:
  - Hadoop Common utilities
  - Avro: A data serialization system with scripting languages.
  - Chukwa: managing large distributed systems.
  - HBase: A scalable, distributed database for large tables.
  - HDFS: A distributed file system.
  - Hive: data summarization and ad hoc querying.
  - MapReduce: distributed processing on compute clusters.
  - Pig: A high-level data-flow language for parallel computation.
  - ZooKeeper: coordination service for distributed applications.

Use Cases for Hadoop

- To aggregate "data exhaust" — messages, posts, blog entries, photos, video clips, maps, web graph
- To give data context — friends networks, social graphs, recommendations, collaborative filtering
- To keep apps running — web logs, system logs, system metrics, database query logs
- To deliver novel mashup services – mobile location data, clickstream data, SKUs, pricing

Hadoop Server Roles



Assumptions and Goals of HDFS

- Hardware Failure
- Streaming Data Access (Best for batch processing)
- Large Data Sets
- Simple Coherency Model (write-once-read-many access model)
- Portability Across Heterogeneous Hardware and Software Platforms

HDFS (Hadoop Distributed File System)

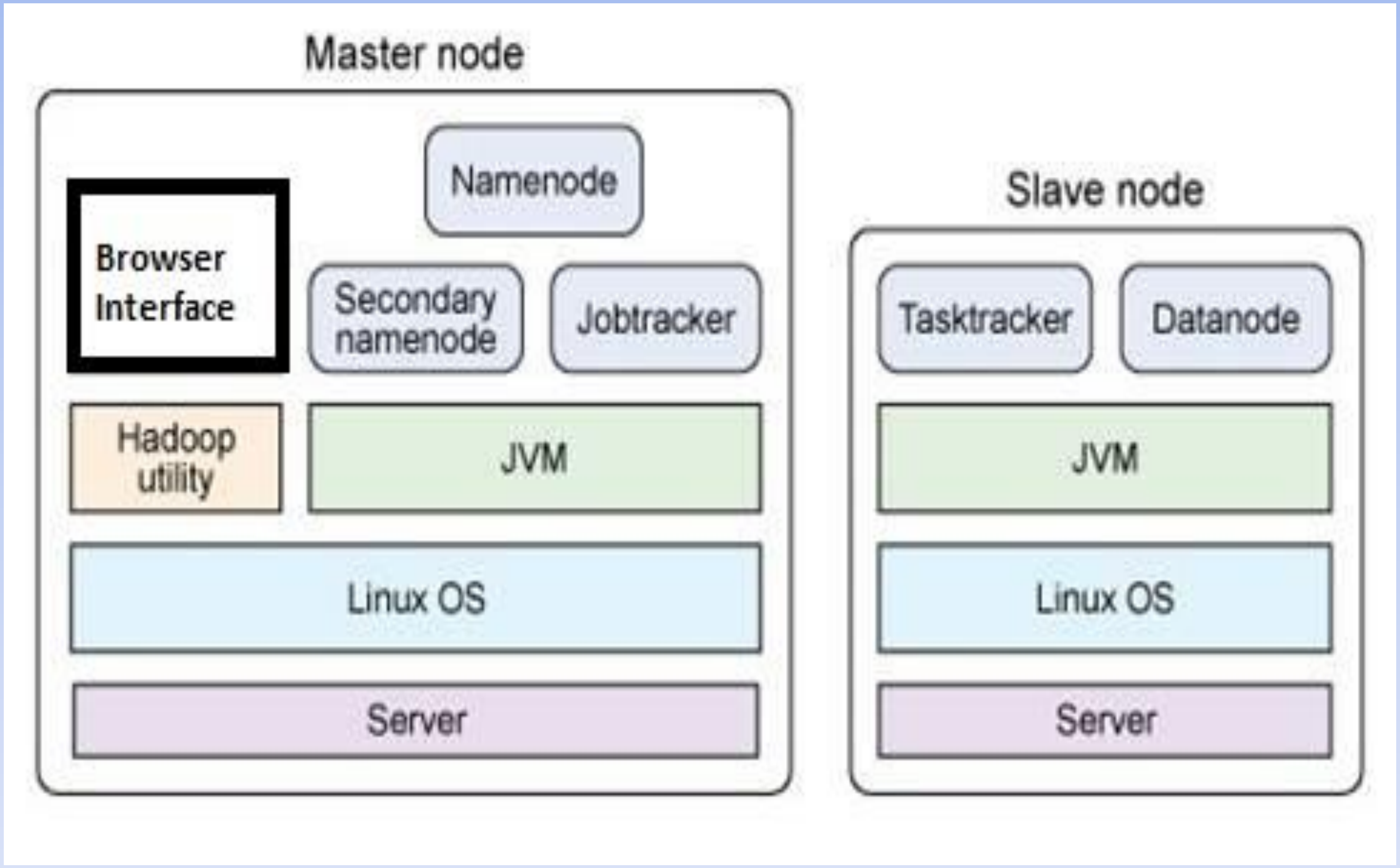
- A distributed file system that provides high-throughput access to application data

- HDFS uses a master/slave architecture in which one device (master) termed as NameNode controls one or more other devices (slaves) termed as DataNode.

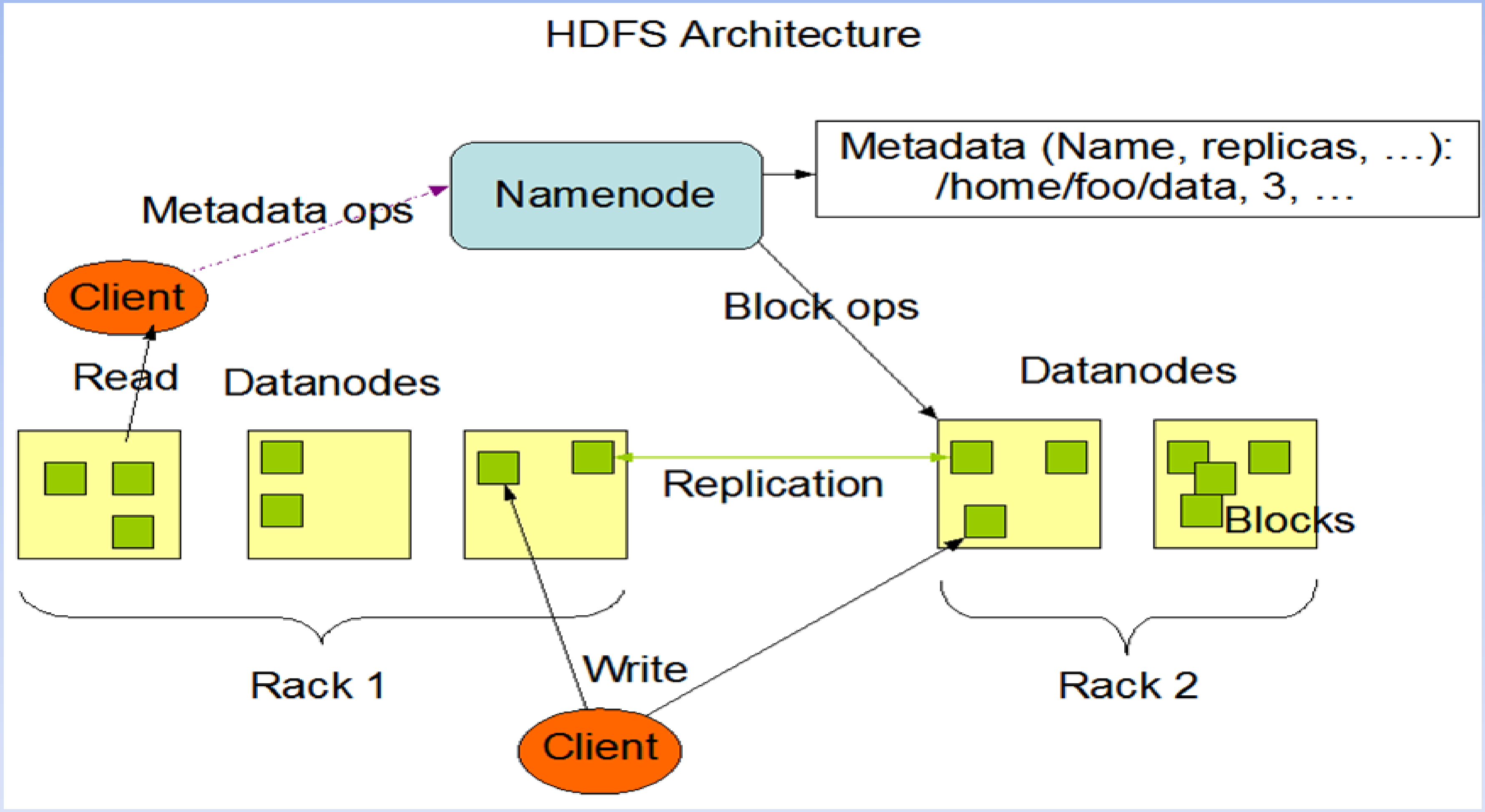
- It breaks Data/Files into small blocks (128 MB each block) and stores on DataNode and each block replicates on other nodes to accomplish fault tolerance.

- NameNode keeps the track of blocks written to the DataNode.

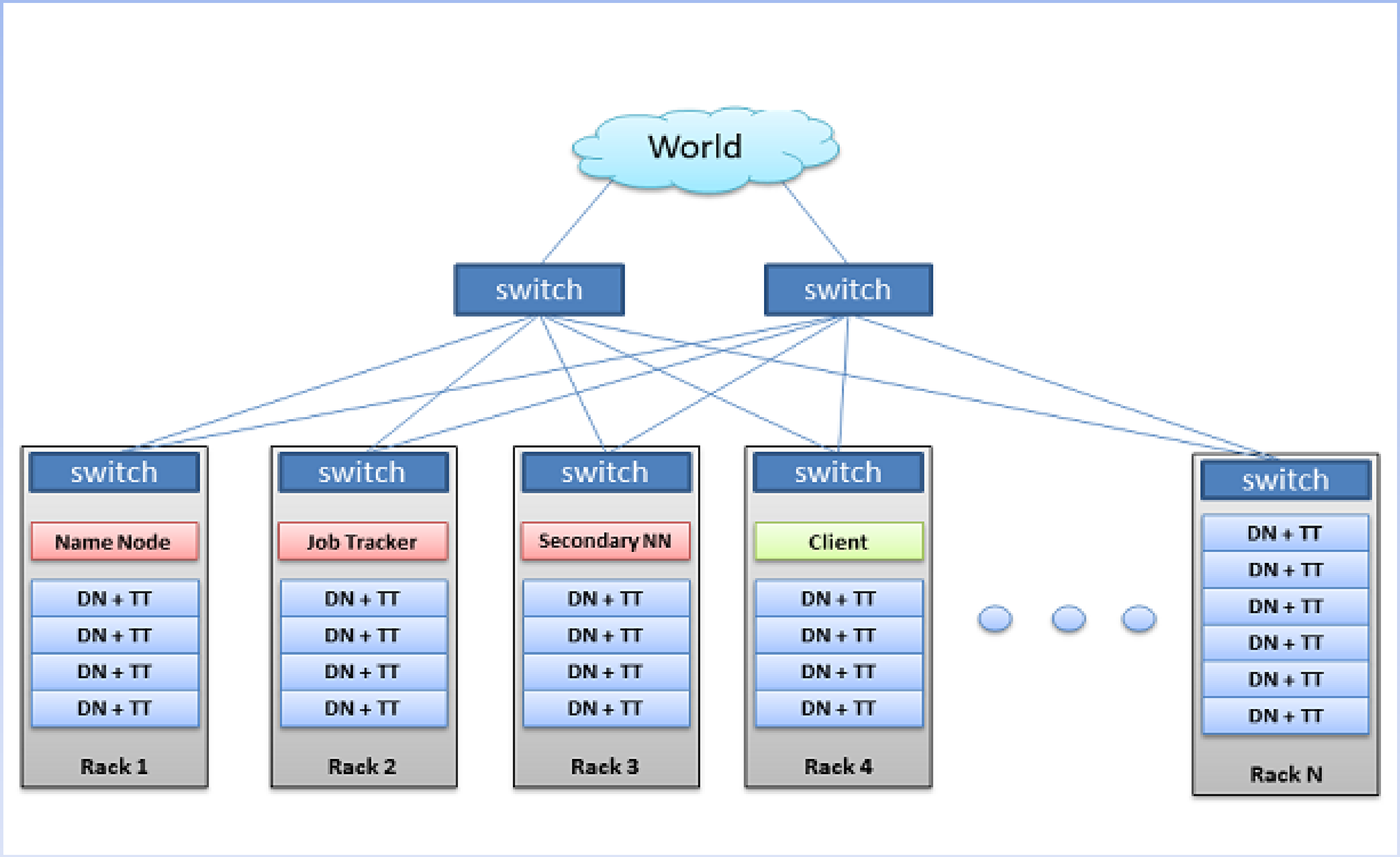
HDFS Cluster Architecture



HDFS Architecture



HDFS Cluster Architecture



HDFS Daemons

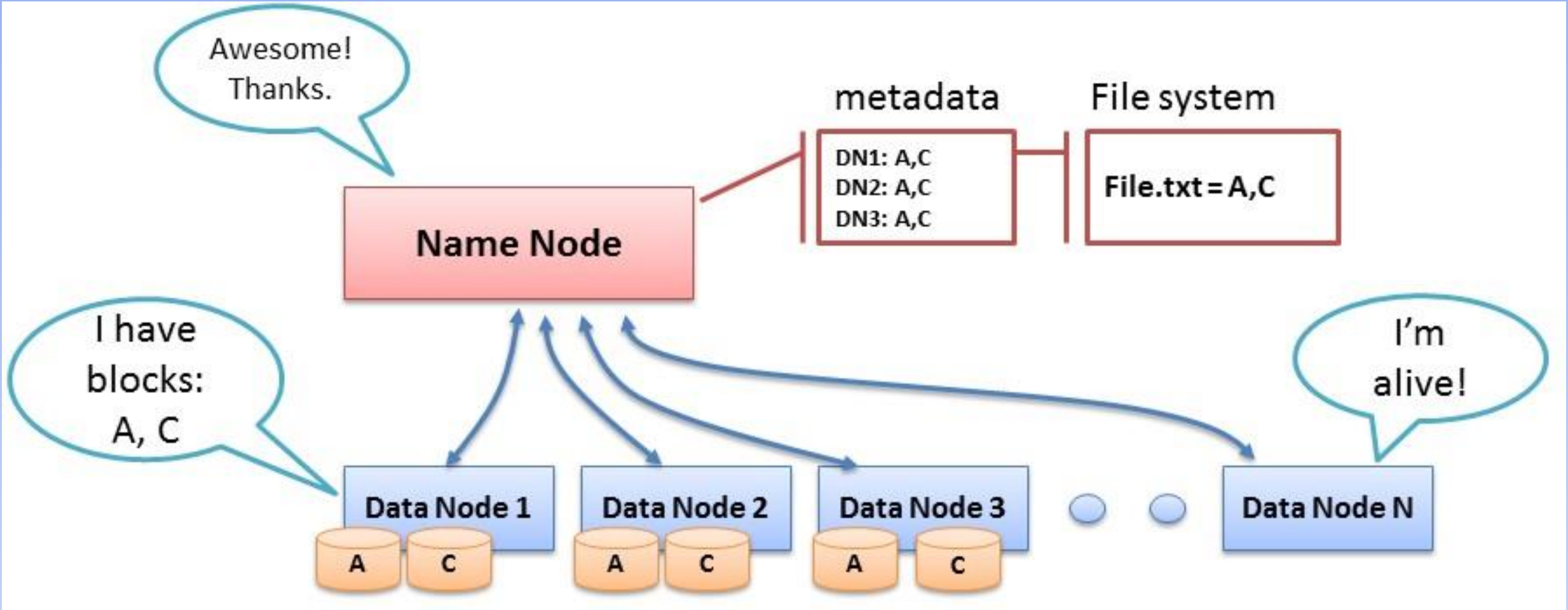
- NAME NODE
- DATA NODE
- SECONDRY NAME NODE

Name Node

- Keeps the metadata of all files/blocks in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself. Kind of block lookup dictionary(index or address book of blocks).

- Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file. The NameNode responds the successful requests by returning a list of relevant DataNode servers where the data lives

Name Node (Contd.)

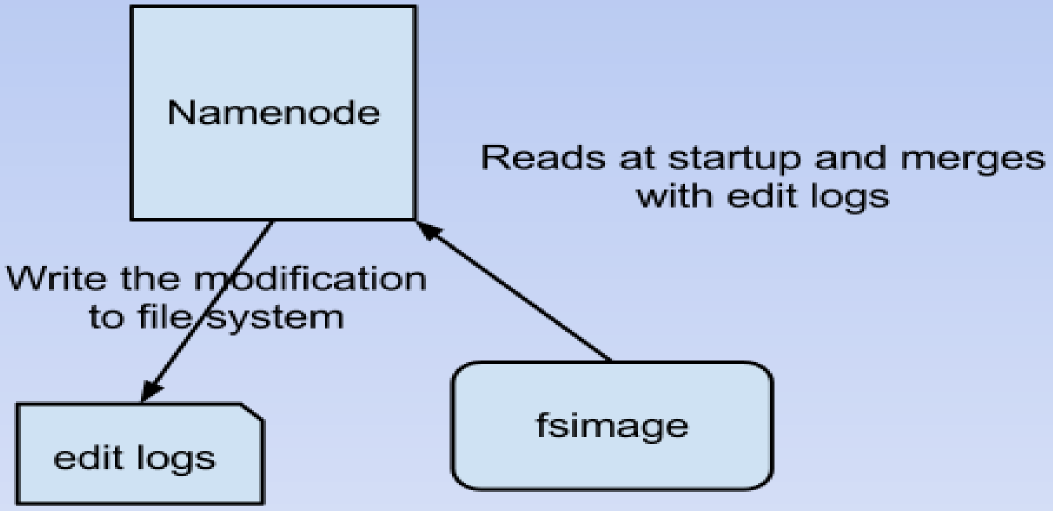


- Data Node sends Heartbeats
- Every 10th heartbeat is a Block report
- Name Node builds metadata from Block reports
- TCP – every 3 seconds
- If Name Node is down, HDFS is down

Name Node (Contd.)

fsimage - Its the snapshot of the filesystem when NameNode started

Edit logs - Its the sequence of changes made to the filesystem after NameNode started


Namenode
Reads at startup and merges with edit logs
Write the modification to file system
edit logs
fsimage

Data Node

- DataNode stores data in the Hadoop Filesystem
- A functional filesystem has more than one DataNode, with data replicated across them
- On startup, a DataNode connects to the NameNode; spinning until that service comes up. It then responds to requests from the NameNode for filesystem operations.
- Client applications can talk directly to a DataNode, once the NameNode has provided the location of the data

Data Replication

- **Why need data replication ?**

- HDFS is designed to handle large scale data in distributed environment
- Hardware or software failure, or network partition exist
- Therefore need replications for those fault tolerance

Replication (Contd.)

- **Replication factor**
  - Decided by users, and can be dynamically tuned.

- **How to Create replications efficiently ?**
  - Replication pipeline: Instead of single machine create replications, a pipe line is applied
  - Machine 1 make replication to machine 2, at the same time machine 2 make the replication to machine 3, etc.

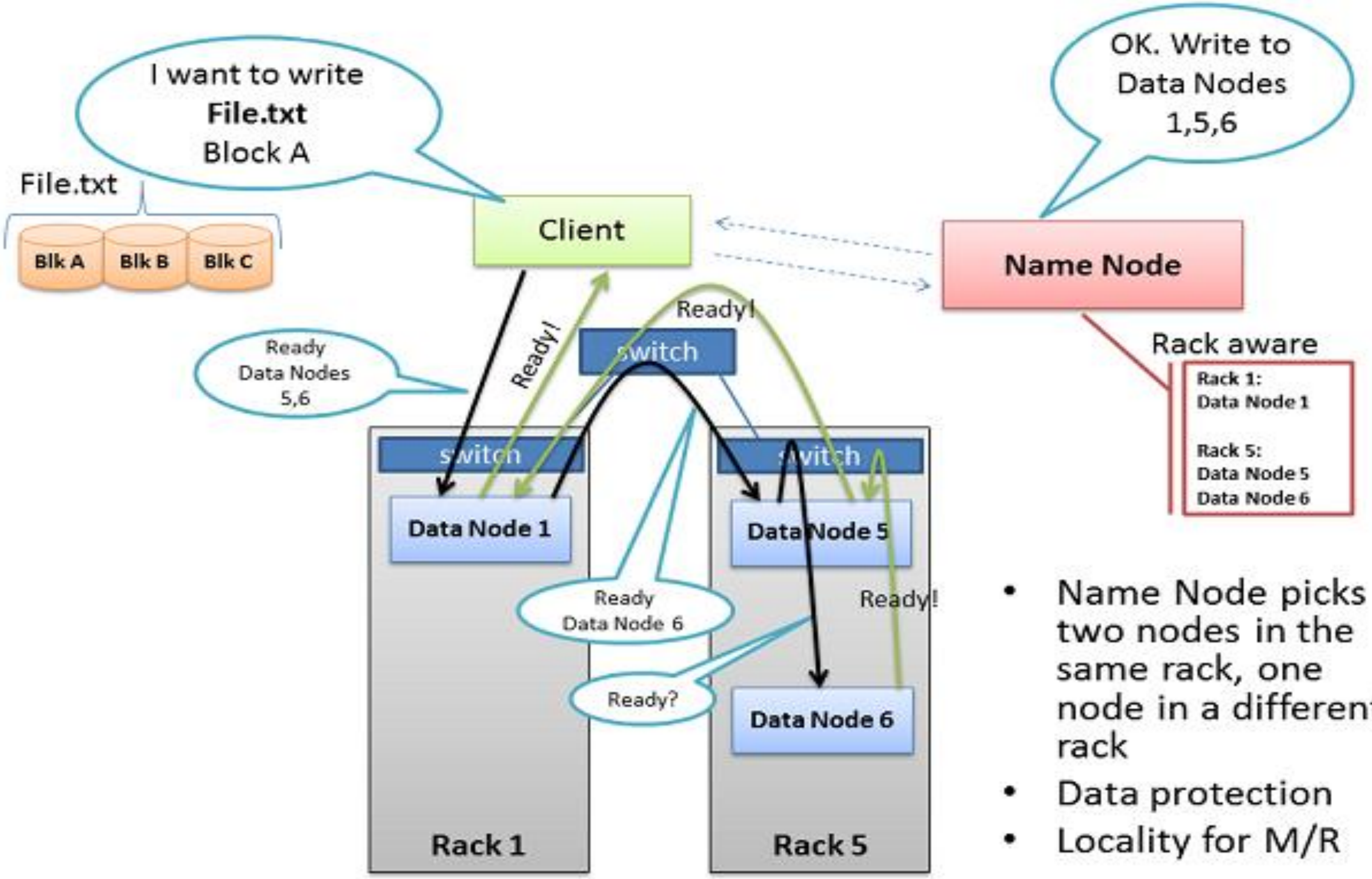
- **Replication placement**
  - High initialization time to create replication to all machines
  - An approximate solution: Only 3 replications

One replication resides in current node

One replication resides in current rack

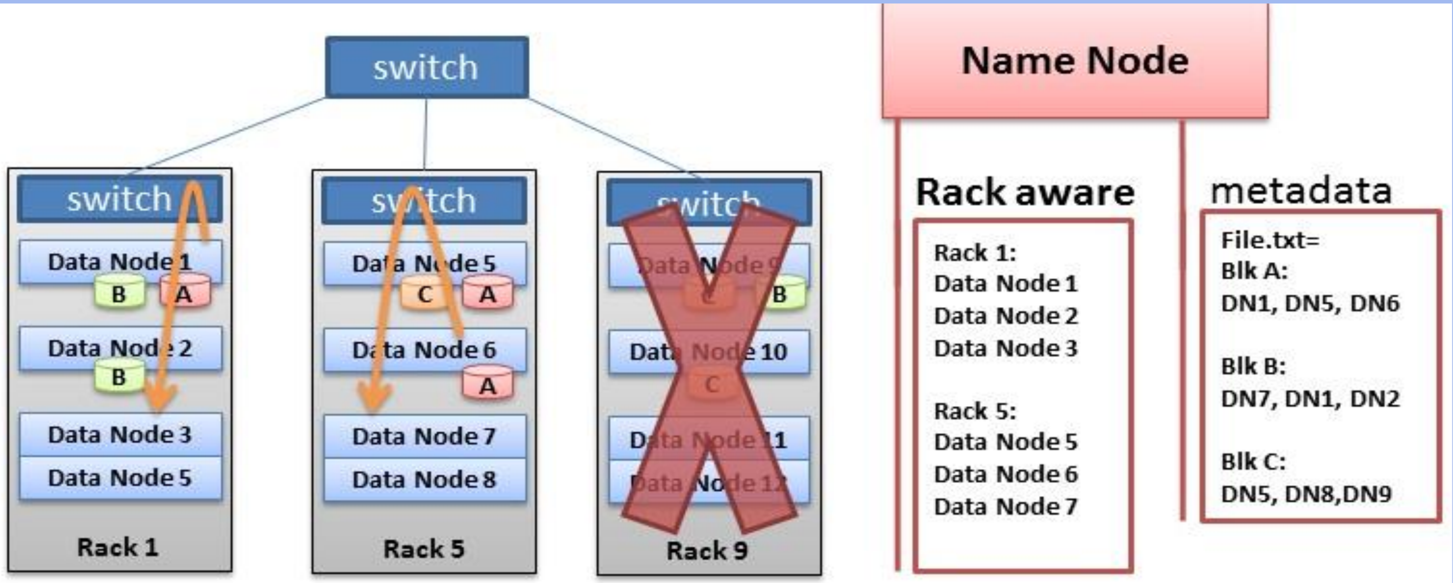
One replication resides in another rack

Replication Pipeline



- Name Node picks two nodes in the same rack, one node in a different rack
- Data protection
- Locality for M/R

Rack Awareness



Rack 1:
Data Node 1
Data Node 2
Data Node 3

Rack 5:
Data Node 5
Data Node 6
Data Node 7

File.txt=
Blk A:
DN1, DN5, DN6

Blk B:
DN7, DN1, DN2

Blk C:
DN5, DN8,DN9

- Never loose all data if entire rack fails
- Keep bulky flows in-rack when possible
- Assumption that in-rack is higher bandwidth, lower latency

Data Node Failure

Data Node Failure Condition

If a data node failed, Name Node could know the blocks it contains, create same replications to other alive nodes, and unregister this dead node

Data Integrity

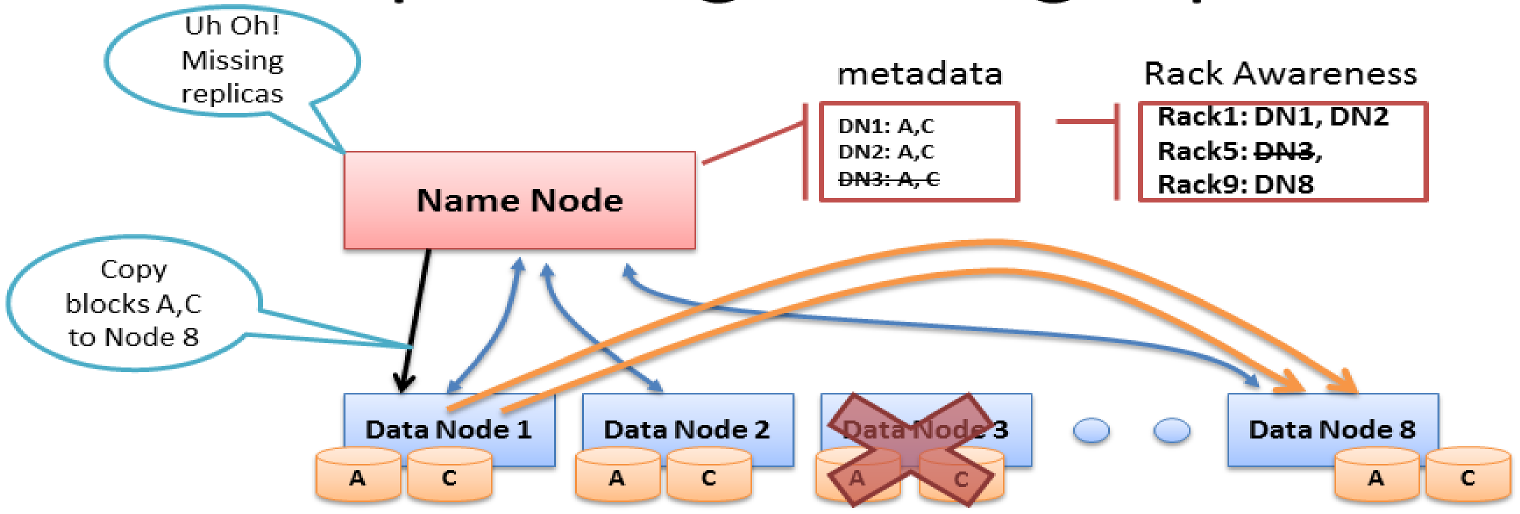
Corruption may occur in network transfer, Hardware failure etc.

Apply checksum checking on the contents of files on HDFS, and store the checksum in HDFS namespace

If checksum is not correct after fetching, drop it and fetch another replication from other machines.

Heartbeats and Re-Replication

Re-replicating missing replicas

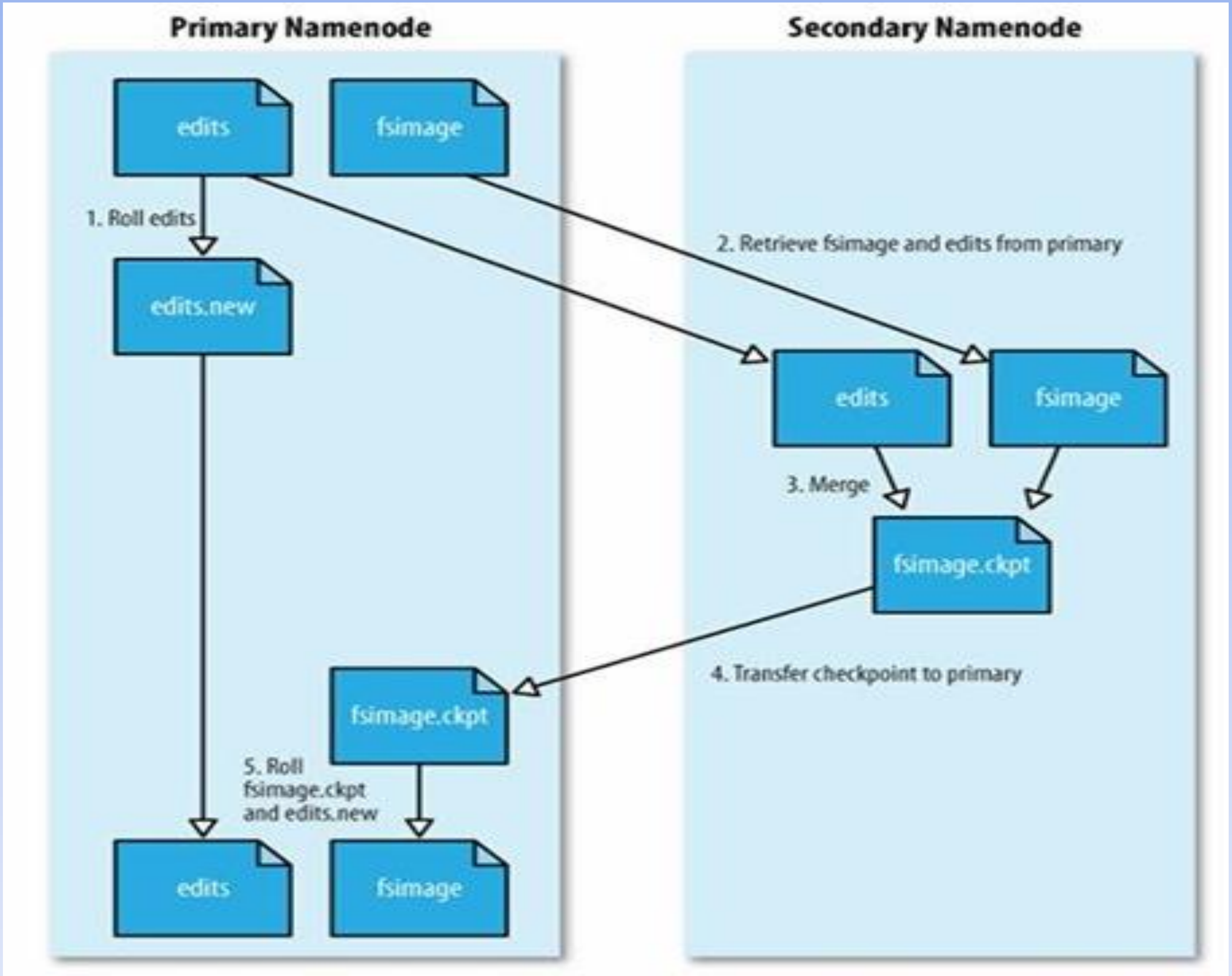


- Missing Heartbeats signify lost Nodes
- Name Node consults metadata, finds affected data
- Name Node consults Rack Awareness script
- Name Node tells a Data Node to re-replicate

Secondary Name Node

- Not a failover NameNode
- The only purpose of the secondary name-node is to perform periodic checkpoints. The secondary name-node periodically downloads current name-node image and edits log files, joins them into new image and uploads the new image back to the (primary and the only) name-node
- Default checkpoint time is one hour. It can be set to one minute on highly busy clusters where lots of write operations are being performed.

Secondary Name Node (Contd.)



Name Node Failure

- NameNode is the single point of failure in the cluster
- If NameNode is down due to software glitch, restart the machine
- If original NameNode can be restored, secondary can re-establish the most current metadata snapshot
- If machine don't come up, metadata for the cluster is irretrievable. In this situation create a new NameNode, use secondary to copy metadata to new primary, restart whole cluster
- Trick : Bring new NameNode up, but use DNS to make cluster believe it's the original one

Apache MapReduce

- A software framework for distributed processing of large data sets
- The framework takes care of scheduling tasks, monitoring them and re-executing any failed tasks.
- It splits the input data set into independent chunks that are processed in a completely parallel manner.
- MapReduce framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically, both the input and the output of the job are stored in a file system.

MapReduce Architecture

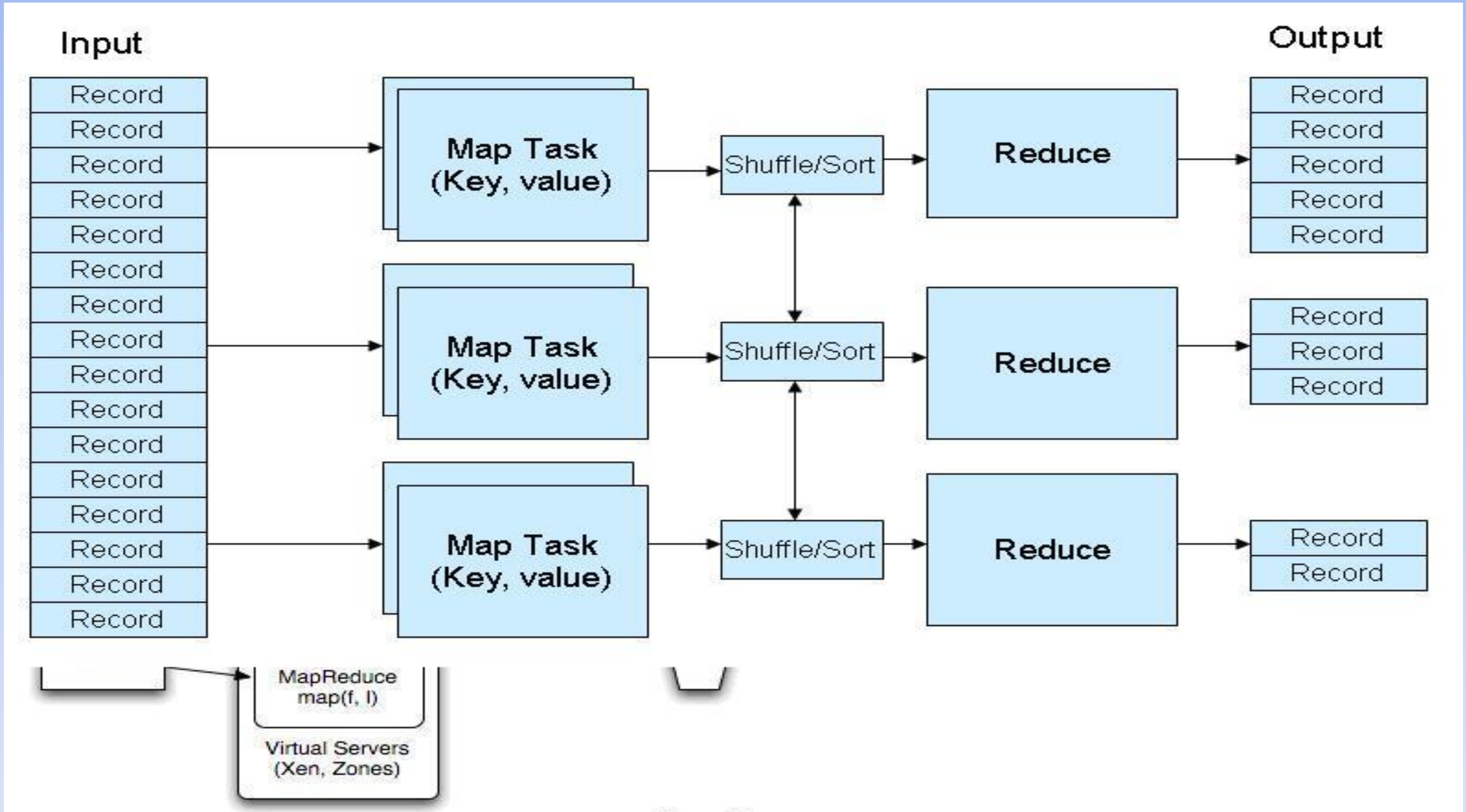
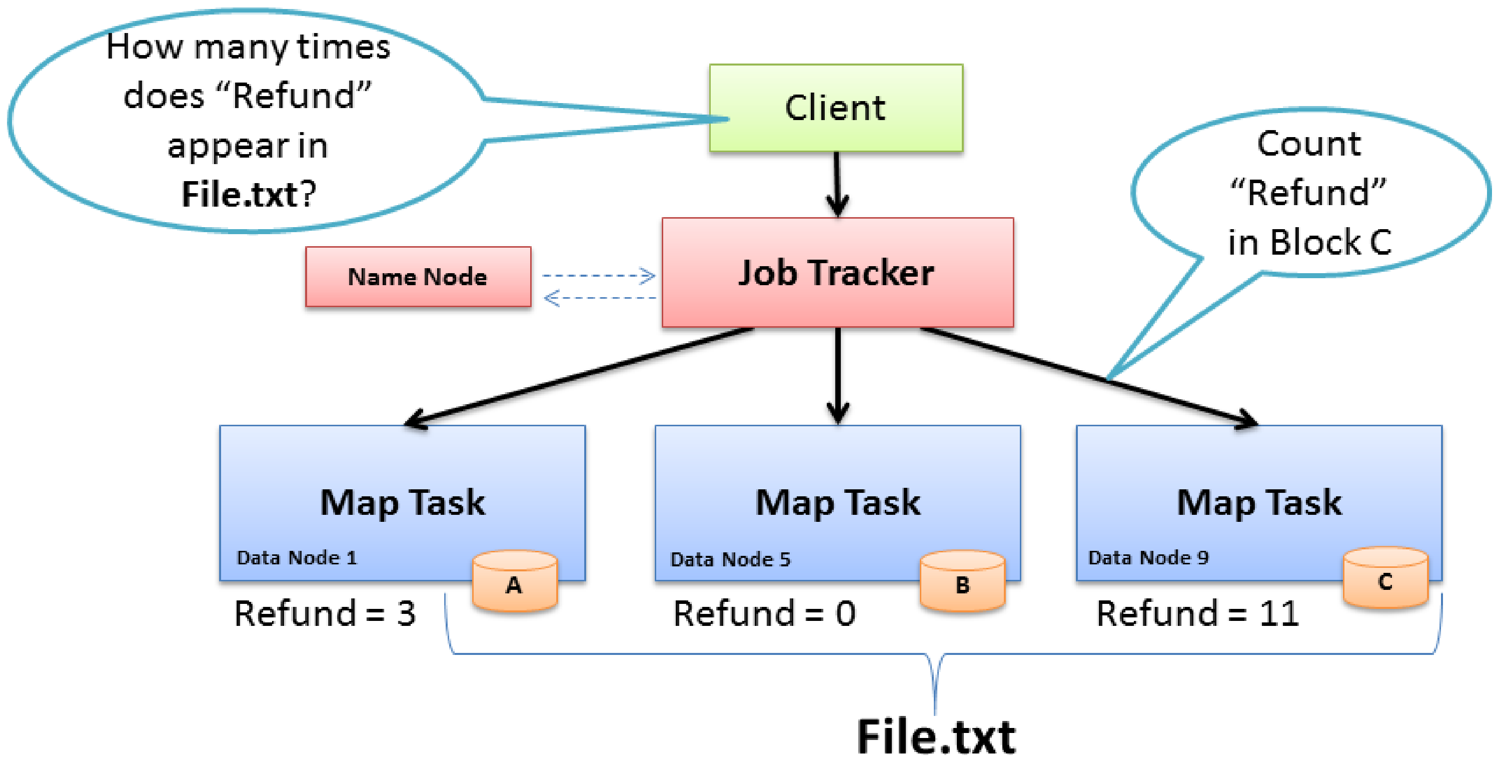


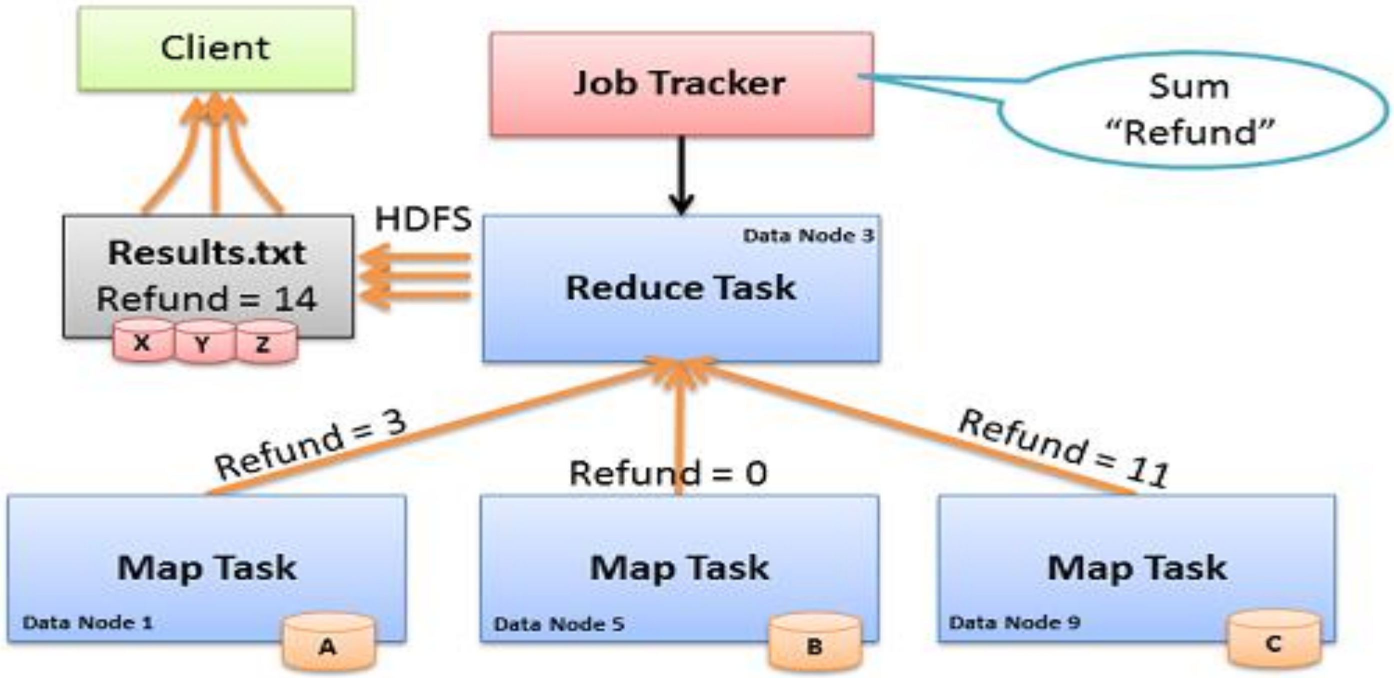
Figure 2

Map Task



- **Map:** "Run this computation on your local data"
- Job Tracker delivers Java code to Nodes with local data

Reduce Task



- **Reduce:** "Run this computation across Map results"
- Map Tasks send output data to Reducer over the network
- Reduce Task data output written to and read from HDFS

MapReduce Dataflow

- An input reader
- A Map function
- A partition function
- A compare function
- A Reduce function
- An output writer

MapReduce Daemons

- JOB TRACKER
- TASK TRACKER

JobTracker

- JobTracker is the daemon service for submitting and tracking MapReduce jobs in Hadoop

- JobTracker performs following actions in Hadoop :

  - It accepts the MapReduce Jobs from client applications
  - Talks to NameNode to determine data location
  - Locates available TaskTracker Node
  - Submits the work to the chosen TaskTracker Node

TaskTracker

- A TaskTracker node accepts map, reduce or shuffle operations from a JobTracker
- Its configured with a set of *slots*, these indicate the number of tasks that it can accept
- JobTracker seeks for the free slot to assign a job
- TaskTracker notifies the JobTracker about job success status.
- TaskTracker also sends the heartbeat signals to the job tracker to ensure its availability, it also reports the no. of available free slots with it.

Hadoop Configuration

- Untar hadoop-*.**.*.tar.gz to your user path

About Version:
The latest stable version 1.0.4 is recommended.

- edit the file conf/hadoop-env.sh to define at least JAVA_HOME to be the root of your Java installation

- edit the files to configure properties:

conf/core-site.xml:

```
<configuration>
    <property>
        <name>
          fs.default.name
        </name>
        <value>
           hdfs://localhost:9000
        </value>
    </property>
</configuration>
```

conf/hdfs-site.xml:

```
<configuration>
    <property>
        <name>
          dfs.replication
        </name>
        <value>
          1
        </value>
    </property>
</configuration>
```

conf/mapred-site.xml:

```
<configuration>
    <property>
        <name>
          mapred.job.tracker
        </name>
        <value>
          localhost:9001
        </value>
    </property>
</configuration>
```

Hadoop Releases

- **1.0.X** - current stable version, 1.0 release
- **1.1.X** - current beta version, 1.1 release
- **2.X.X** - current alpha version
- **0.23.X** - simmilar to 2.X.X but missing NN HA.
- **0.22.X** - does not include security
- **0.20.203.X** - old legacy stable version
- **0.20.X** - old legacy version

HDFS Access Methods

- Java API (For applications)

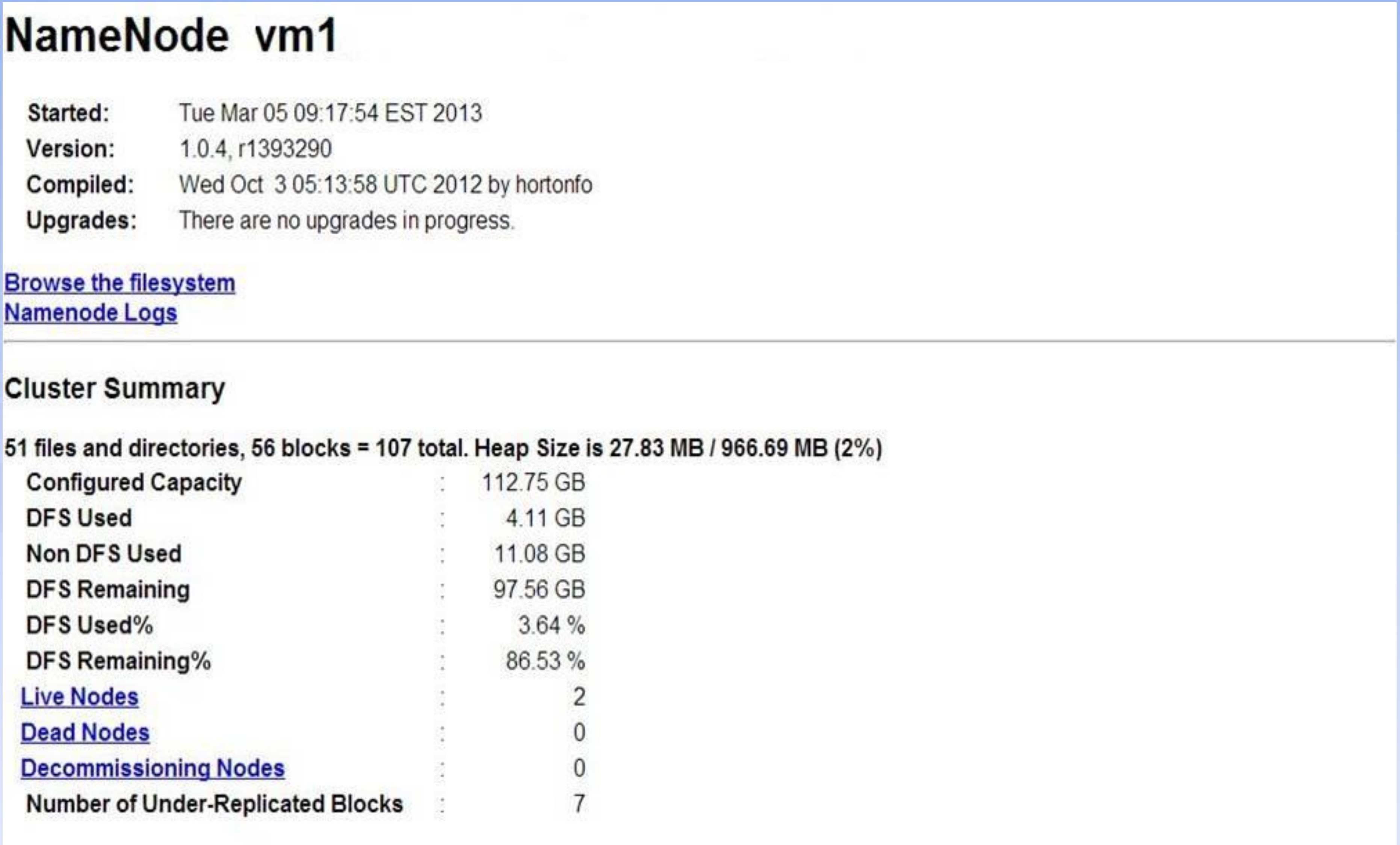
- Browser Interface (Next Slide)

- Hadoop FS Shell
  - Formatting filesystem with HDFS
    - bin/hadoop namenode -format
  - To add a directory
    - bin/hadoop dfs –mkdir abc
  - To list a directory
    - bin/hadoop dfs -ls /
  - To display content of a file
    - bin/hadoop dfs -cat filename

Hadoop Browser Interfaces

- MapReduce Job Tracker Web Interface
  http://localhost:50030/
- Task Tracker Web Interface
  http://localhost:50060/
- HDFS Name Node Web Interface
  http://localhost:50070/

Name Node Interface



NameNode vm1

Started:	Tue Mar 05 09:17:54 EST 2013
Version:	1.0.4, r1393290
Compiled:	Wed Oct 3 05:13:58 UTC 2012 by hortonfo
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

51 files and directories, 56 blocks = 107 total. Heap Size is 27.83 MB / 966.69 MB (2%)

Configured Capacity	:	112.75 GB
DFS Used	:	4.11 GB
Non DFS Used	:	11.08 GB
DFS Remaining	:	97.56 GB
DFS Used%	:	3.64 %
DFS Remaining%	:	86.53 %
Live Nodes	:	2
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	7

References

- http://hadoop.apache.org/
- http://wiki.apache.org/hadoop/
- http://hadoop.apache.org/core/docs/current/hdfs_design.html
- http://wiki.apache.org/hadoop/FAQ

Questions?

- Write your queries to mkjoshi@expresskcs.com